AF 3700

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Pelegri-Llopart et al.	Attorney Docket No.: SUN1P254/P4195
Application No.: 09/471,072	Examiner: Kiss, Eric B.
Filed: December 21, 1999	Group: 2122
Title: MECHANISM FOR AUTOMATIC SYNCHRONIZATION OF SCRIPTING VARIABLES	Confirmation No. 6969

CERTIFICATE OF MAILING
I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first-class mail on November 3, 2004 in an envelope addressed to the Commissioner for Patents, Mail Stop Appeal Brief-Patents, P.O. Box 1450 Alexandria, VA 22313-1450.

Signed: _____
Agnes Spence

AMENDED APPEAL BRIEF TRANSMITTAL
(37 CFR 41.37)

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In response to the Notification of Non-Compliant Appeal Brief attached is the amended Appeal Brief.

This application is on behalf of Large Entity.

Applicant previously paid the Appeal Brief fee in the amount of $330.00 with the previously filed Appeal Brief. However, if the Commissioner believes that such fees or extension fees are required, the Applicant hereby petition that such an extension be granted and authorize the Commissioner to charge the required fees to Deposit Account No. 500388 (Order No. SUN1P254).

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

Fredrik Mollborn
Reg. No. 48,587

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF APPEALS

EX PARTE EDUARDO PELEGRI-LLOPART ET AL.

Application for Patent

Filed December 21, 1999

Application No. 09/471,072

FOR:

MECHANISM FOR AUTOMATIC SYNCHRONIZATION OF SCRIPTING VARIABLES

AMENDED APPEAL BRIEF

CERTIFICATE OF MAILING
I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first-class mail on November 3, 2004 in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450.

Signed: _____
Agnes Spence

BEYER WEAVER & THOMAS, LLP
Attorneys for Applicant

11/08/2004 SSESHE1 00000078 500388 09471072
01 FC:1402 340.00 DA

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest is Sun Microsystems, Inc., the assignee of the present application.

II. RELATED APPEALS AND INTERFERENCES

The undersigned is not aware of any related appeals and/or interferences.

III. STATUS OF CLAIMS

There are a total of 12 claims pending in this application (claims 1, 4, 6-8, 10, and 13-18). Claims 2-3, 5, 9 and 11-12 have been canceled during prosecution.

Claims 1, 4, 7-8, 13-15 and 17-18 stand rejected under 35 USC §102(a) as being anticipated by the ColdFusion 4.0 software product, as evidenced by the ColdFusion Documentation files, including "Getting started with ColdFusion" (hereinafter CF Getting Started); "Developing Web Applications with ColdFusion" (hereinafter CF Web); "Advanced ColdFusion Development" (hereinafter CF Advanced); "ColdFusion 4.0 Documentation Update" (hereinafter CF Update); and "ColdFusion Quick Reference Guide" (hereinafter CF Guide), and such a product hereinafter ColdFusion 4.0.

Claims 6, 10 and 16 stand rejected under 35 USC §103(a) as being unpatentable over the ColdFusion 4.0 software product as applied to claims 1, 7 and 13 in the 35 USC §102(a) rejection.

Claims 1, 4, 6-8, 10, and 13-18 are appealed in this brief.

IV. STATUS OF AMENDMENTS

The applicants filed an amendment after the final rejection by the Examiner dated January 5, 2004. This amendment was labeled Amendment After Final, and was filed April 5, 2004. As indicated in the Advisory Action dated May 3, 2004, the Amendment After Final has been entered but the Examiner stated that the Applicant's response to the final rejection had been considered but was not deemed to place the application in condition for allowance.

The applicants filed a supplemental amendment along with the original filing of this appeal brief on September 23, 2004. The supplemental amendment addressed the objections to the specification and adopted the Examiner's suggestions with regards to the use of the trademarks JAVA™ and JAVASERVER™. This supplemental amendment has not yet been entered.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Independent claim 1

The present invention relates generally to the field of computer software. More specifically, the present invention, as defined in claim 1, relates to a computer system for automatic synchronization of scripting variables between a page that includes action tags and a tag library.

The system of claim 1 includes "a page suitable for building an application with dynamic web content, the page including one or more action tags that are provided as text in a mark-up language." One example of such a page is a JAVASERVER™ Page. JAVASERVER™ Pages is the JAVA™ platform technology provided by Sun Microsystems, Inc. for building applications containing dynamic web content, such as HTML, DHTML, XHTML, and XML. A JAVASERVER™ Page (JSP) is a text-based document that describes how to process a request in order to create a response. The description in the JSP inter-mixes template data, which is commonly fragments of a structured document (such as a HTML document, a DHTML document, an XHTML document or an XML document), with dynamic actions that are described with scripting elements and/or server-side action tags (Specification, paragraph **[0003]**).

Typically, a JSP contains, among other things, scripting variables and action tags. A scripting variable is a variable that is used by a script. Scripts are lists of commands that can be executed without user interaction. Action tags are commands that can define new scripting variables or update existing scripting variables (Specification, paragraph **[0029]**, FIG. 4).

A JSP is executed by a JSP container on a web server, or on a web enabled application server. The JSP container delivers requests from a client to the JSP and delivers responses from the JSP back to the client (Specification, paragraph **[0005]**). The first time a JSP is accessed, the JSP is compiled, that is, the high-level programming language code in which the JSP is written is translated into JAVA™ code that runs in the JSP container on the server. This is referred to as "translation time." The compiled JAVA™ code on the server is subsequently executed whenever a request is received, that is, at "run time" (Specification, paragraph **[0007]**).

The system of claim 1 further includes "a tag library." The tag library contains available action tags (i.e., available actions that can be performed) that JSP authors can use when creating JSPs. Analogous to conventional libraries, the action tags in the tag library is a collection of precompiled action tags that are stored in an object format. Tag libraries are particularly useful because they remove the need to explicitly link the action tags to every application that uses the

action tags. Instead, during compilation the linker automatically looks in the available action tag libraries for action tags that the linker does not find elsewhere. The action tag library in accordance with the invention includes a tag library descriptor (TLD). For each action tag, the TLD includes a tag handler class (i.e. a request time object of the action tag), and a TagExtraInfo class (Specification, paragraph **[0030]**), which will be described in further detail below.

The system further includes:

"a translator suitable for translating the action tags from the mark-up language to an executable programming code that is executed at runtime to perform actions intended by the action tags; "

and

"a TagExtraInfo object for each action tag in the page, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method returning, at translation time, information that includes a list of available scripting variables, and a variable type and scope associated with each scripting variable that is defined or modified by its associated action tag, thereby allowing the translator at translation time to use the information provided by the method to generate code that when executed at runtime will assign each of the scripting variables with appropriate runtime values with respect to the type and scope of each of the scripting variables;"

Thus, the translator translates action tags at translation time from the mark-up language into an executable programming code that can be executed at runtime. Specifically, at translation-time, the JSP with the action tags is translated into a servlet class, that is, a small program that runs on the web server. Each action tag in the JSP has a corresponding a TagExtraInfo object that contains information about the attributes of the object, and maintains a list of scripting variables introduced or modified by the associated action. In the process of translation, a translator asks for the list of scripting variables from the TagExtraInfo object which are affected by a given action, and the method of the TagExtraInfo object returns at translation time an array of objects that describe the run-time effect of the action. If a new variable has been defined, the translator may need to provide a new declaration statement. (Specification, paragraphs **[0030]-[0031]**).

The system of claim 1 also includes:

"a pageContext object for the page, the pageContext object including a runtime mapping of at least one scripting variable in the list of available scripting variables to a runtime value that is represented or can be represented in the tag library, "

and

"a tag handler that creates at runtime one or more objects that the page requires, the tag handler further operating to store the one or more created objects into the pageContext object; thereby allowing the one or more objects to be retrieved at runtime when the generated code is executed, the one or more objects being assigned at runtime to

each of the scripting variables in the list of scripting variables that is returned by the method at translation time."

Thus, at run time, the tag handler references a pageContext object, which provides key values for specified variable names. The pageContext object is created when the JSP is executed and points to an object associated with a given variable name, thereby associating a scripting variable with a runtime value (Specification, paragraph **[0031]**). At run time, the executable programming code (that was generated by the translator at translation time) accesses the pageContext object according to some contract that is described by a combination of specification-defined conventions plus the information provided by the TagExtraInfo object, looks for the name of a variable and assigns the variable's value to the scripting variable. (Specification, paragraph **[0032]**).

As can be seen from the above description, the invention combines an explicit run-time representation of the context (i.e., the pageContext object) with translation-time information (i.e., the TagExtraInfo class) for the affected scripting variables. The action tags use the run-time context (i.e. the pageContext object) to modify and/or create objects. The JSP uses the translation-time information (i.e. the TagExtraInfo class) to automatically synchronize the scripting variables whenever the JSP code is accessed. Expressed differently, the TagExtraInfo object contains information about which variables will be modified, and the scripting language knows how to do the modification. By providing a mechanism for the tag library and the JSP container (translator) to share this information, the invention facilitates the use of many different scripting languages for JSPs (Specification, paragraph **[0034]**). Furthermore, tag libraries can be standardized, so that the same tag library can be supported by different web browsers, regardless of how the web browsers support scripting (Specification, paragraph **[0011]**).

Independent claim 13

Independent claim 13 defines a computer readable medium including computer program code for automatically synchronizing scripting variables between a page including one or more action tags and a tag library. Claim 13 is a *Beauregard* claim with claim features that are analogous to the claim features that have been described above with respect to claim 1. For this reason, no further explanation of the features of claim 13 will be provided in this section.

Independent claim 7

Independent claim 7 defines a method for automatically synchronizing a scripting variable between a page including one or more action tags and a tag library. Claim 7 recites:

> A method for automatically synchronizing scripting variable between a page including one or more action tags and a tag library, the page suitable for building an application with dynamic web content, the one or more action tags being provided as text in a mark-up language which are translated at translation time to an executable code that is executed at runtime to synchronize the scripting variables at runtime, the method comprising:
>
> instantiating, by a translator at translation time, for each action tag a TagExtraInfo object, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method capable of returning at translation time information that includes a list of available scripting variables and respectively associated variable types and scopes for each of the scripting variables in the list of available scripting variables, each of the scripting variables being defined or modified by an associated action tag;
>
> invoking the method by a translator at translation time, wherein the invoking operates to pass a list of attributes associated with the one or more action tags;
>
> receiving, as a result of the invoking of the method, a collection of returned TagExtraInfo objects, wherein each of the returned TagExtraInfo objects includes an available scripting variable, its variable type, and its scope in the page;
>
> generating by the translator, based on the returned TagExtraInfo objects, executable code that is executed at runtime, wherein the executable code accesses at runtime data that will be stored in a pageContext object at runtime, the runtime data including appropriate runtime values for each of the available scripting variables;
>
> storing at runtime, into the pageContext object, by a tag handler at runtime, one or more objects that the page requires, thereby allowing the objects for the one or more objects to be retrieved at runtime and be assigned at runtime to the list of scripting variables; and
>
> executing, at run time, the code generated by the translator to assign appropriate runtime values which are stored in the pageContext object to each of the scripting variables in the returned TagExtraInfo objects, thereby allowing the runtime values to be retrieved and assigned at runtime to each of the available scripting variables in the collection of returned TagExtraInfo objects.

As can be seen, the limitations of claim 7 are similar to the limitations of the independent system claim 1, discussed above, so the explanation below is focused on a feature of claim 7 that

has not been explained above in the context of claim 1. Claim 7 includes the step:

> invoking the method by a translator at translation time, wherein the invoking operates to pass a list of attributes associated with the one or more action tags;

Thus, the translator invokes the method of the TagExtraInfo object, and in the invocation a list of attributes associated with the one or more action tags is passed. There are several types of attributes. For example, an action tag can define one or more objects, and an id attribute may be used to describe the "main" object (if any) defined by the tag. Another example is a scope attribute, which can be used to describe where to introduce the object (Specification, paragraph **[0037]**).

Dependent claims 4 and 14

Claims 4 and 14 depend directly from independent claims 1 and 13, respectively, and, define the following features of the TagExtraInfo object:

> "the TagExtraInfo object comprises:
> a valid object name for each variable;
> a type for each variable; and
> a scope parameter that specifies a variable's scope relative to the page. "

That is, each variable in the TagExtraInfo object has a valid object name, a type, and a scope parameter (Specification, paragraph **[0030]**). This allows the translator at translation time to obtain all the necessary information about the all the variables that are affected at runtime by the associated action tag, and to generate code to be executed at runtime based on the information in the TagExtraInfo object.

Dependent claim 8

Claim 8 depends directly from independent claims 7 and defines the following features of the TagExtraInfo object:

> "the TagExtraInfo object comprises:
> a valid object name for each variable;

a type for each variable; and
a scope parameter that specifies a variable's scope relative to the page. "

That is, each variable in the TagExtraInfo object has a valid object name, a type, and a scope parameter (Specification, paragraph **[0030]**). This allows the translator at translation time to obtain all the necessary information about the all the variables that are affected at runtime by the associated action tag, and to generate code to be executed at runtime based on the information in the TagExtraInfo object.

Dependent claims 15 and 17

Claims 15 and 17 depend directly from independent claims 13 and 1, respectively, and recite:

"the page is converted to a first programming code which is different than a second programming code that is used to implement the tag library."

Thus, claims 15 and 17 explicitly state that the page (for example, the JSP) is converted to a programming code (for example, scripting code that can be executed at runtime), and that this code is different from a programming code in which the tag library is implemented. That is, there is no need for the tag library and the JSP to be written in the same language (which would have allowed the variables to be synchronized more easily). Instead, the present invention provides a more general solution by exposing the variables in an action tag at translation time through the TagExtraInfo class, and using a pageContext object to map the variables to values at run time, as discussed above (Specification, paragraph **[0033]**).

Dependent claim 18

Claim 18 depends directly from independent claim 7, and recites:

"the page is converted to a first programming code which is different than a second programming code that is used to implement the tag library."

Thus, claim 18 explicitly states that the page (for example, the JSP) is converted to a programming code (for example, scripting code that can be executed at runtime), and that this code is different from a programming code in which the tag library is implemented. That is, there is no need for the tag library and the JSP to be written in the same language (which would

have allowed the variables to be synchronized more easily). Instead, the present invention provides a more general solution by exposing the variables in an action tag at translation time through the TagExtraInfo class, and using a pageContext object to map the variables to values at run time, as discussed above (Specification, paragraph **[0033]**).

Claims 6 and 16

Claims 6 and 16 depend directly from independent claims 1 and 13, respectively, and recite:

> "the page is executed on a server that implements a container, and the page is converted to a platform independent code that is executed on the server."

Thus, the first part of claims 6 and 16 (that is, "the page is executed on a server that implements a container") specifies features of the JAVA™ technology. A JSP is executed by a JSP container, which is installed on a Web server, or on a Web enabled application server. The JSP container delivers requests from a client to a JSP page and responses from the JSP page to the client. JSP pages may be implemented using a JSP translation or compilation phase that is performed only once, followed by a request processing phase that is performed once per request. The translation phase creates a JSP page implementation class that implements a servlet interface (Specification, paragraph **[0005]**).

The second part of claims 6 and 16 (that is, "the page is converted to a platform independent code that is executed on the server") specifies that the converted page (e.g. the JSP) is executed on the server, and that the representation of the converted page is platform independent (such as a JAVA™ servlet) (Specification, paragraph **[0031]**).

Claim 10

Claim 10 depends directly from independent claim 7 and recites:

> "the page is executed on a server that implements a container, and the page is converted to a platform independent code that is executed on the server."

Thus, the first part of claim 10 (that is, "the page is executed on a server that implements a container") specifies features of the JAVA™ technology. A JSP is executed by a JSP container, which is installed on a Web server, or on a Web enabled application server. The JSP container delivers requests from a client to a JSP page and responses from the JSP page to the client. JSP pages may be implemented using a JSP translation or compilation phase that is performed only once, followed by a request processing phase that is performed once per request. The translation phase creates a JSP page implementation class that implements a servlet interface (Specification, paragraph **[0005]**).

The second part of claim 10 (that is, "the page is converted to a platform independent code that is executed on the server") specifies that the converted page (e.g. the JSP) is executed on the server, and that the representation of the converted page is platform independent (such as a JAVA™ servlet) (Specification, paragraph **[0031]**).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

(a) Claims 1, 4, 7-8, 13-15 and 17-18 stand rejected under 35 USC §102(a) as being anticipated by the ColdFusion 4.0 software product, as evidenced by the ColdFusion Documentation files, including "Getting started with ColdFusion" (hereinafter CF Getting Started); "Developing Web Applications with ColdFusion" (hereinafter CF Web); "Advanced ColdFusion Development" (hereinafter CF Advanced); "ColdFusion 4.0 Documentation Update" (hereinafter CF Update); and "ColdFusion Quick Reference Guide" (hereinafter CF Guide), and such a product hereinafter ColdFusion 4.0.

(b) Claims 6, 10 and 16 stand rejected under 35 USC §103(a) as being unpatentable over the ColdFusion 4.0 software product as applied to claims 1, 7 and 13 in the 35 USC §102(a) rejection.

VII. ARGUMENT

A) The rejection of claims 1, 4, 7-8, 13-15 and 17-18 under 35 U.S.C. §102(a).

1. Independent claims 1 and 13

Claims 1 and 13 pertain to a computer system and a computer readable medium, respectively, for automatic synchronization of scripting variables between a page including action tags and a tag library. Both claims 1 and 13 are rejected under 35 U.S.C. §102(a) as being anticipated by the ColdFusion 4.0 software product.

(a) The cited art does not teach automatic synchronization of scripting variables between a page including action tags and a tag library.

The respective preambles of claims 1 and 13 recite:

> "A computer system for automatic synchronization of scripting variables between a page including action tags and a library..."

and

> "A computer readable media including computer program code for automatically synchronizing scripting variables between a page including one or more action tags and a tag library..."

As was described above, there is a need to automatically synchronize scripting variables between a JSP and a tag library, such as an action tag library. Claims 1 and 13 are directed to a system and a computer readable medium, respectively, for accomplishing this, as can be seen from the preambles of claims 1 and 13. The Examiner has pointed out in numerous office actions, in particular in the latest Advisory Action mailed on May 3, 2004, that the *CFSET* object of ColdFusion 4.0 includes a mapping of scripting variables to values (citing the first example given on page 15 of the CF Web reference, which shows a mapping of the variable "FirstName" to the value "Jack"). The applicants respectfully disagree for at least the following reasons.

First, claims 1 and 13 require synchronization of scripting variables, which is a particular type of variable. As was stated above, a scripting variable is a variable that is used by a script, that is, a list of commands that can be executed without user interaction. ColdFusion supports a wide range of variable types, which are listed in the "Kinds of Variables" section starting on page

27 of CF Web. No variable type that is described in the "Kinds of Variables" section of CF Web is equivalent to the scripting variables of the applicants' invention.

Second, as can be seen from the preambles, the scripting variables are synchronized, that is, automatically updated, between a page including one or more action tags (for example, a JSP) and a tag library (for example, an action tag library). The applicants respectfully point out that the Examiner has not shown any scripting variables being synchronized anywhere in ColdFusion. Instead, the Examiner just makes a general allegation that the *CFSET* object of ColdFusion includes a mapping of scripting variables to values (see, for example, the latest Advisory Action mailed on May 3, 2004, citing the first example given on page 15 of the CF Web reference, which shows a mapping of the variable "FirstName" to the value "Jack"). Furthermore, the applicants have failed to find any tag libraries, or equivalents thereof, in ColdFusion. Nor does the Examiner's office actions provide any guidance about where such tag libraries can be found in ColdFusion. The Examiner appears to interpret "tag library" merely as a "collection of tags" (see, for example, the last paragraph on page 7 of the Final Office action mailed January 5, 2004). As was discussed above, a tag library is not only a collection of tags, but is a collection of precompiled tags that are stored in an object format and thereby removes the need to explicitly link the tags to every application that uses the action tags. Moreover, even if one were to agree with the Examiner's interpretation of a tag library to merely be a "collection of tags," the Examiner has failed to provide a reference as to where such a "collection of tags" can be found in the cited ColdFusion documentation.

(b) The cited art does not teach translation of the page into an executable programming code

Claims 1 and 13 recite:

> "a translator suitable for translating the action tags from the mark-up language to an executable programming code that is executed at runtime to perform actions intended by the action tags"

The Examiner has asserted that the ColdFusion 4.0 software product teaches a translator suitable for translating an action tag from a mark-up language to an executable programming code that is executed at runtime to perform actions intended by the action tags (Final Office Action, page 7, paragraph 10, citing CF Web, page 7, first three paragraphs). The applicants respectfully disagree. The ColdFusion software product does not translate action tags from the

mark-up language into an executable programming code that is executed at runtime. Instead, in the ColdFusion 4.0 software application, application pages are created to capture data and provide output. These application pages can contain "ColdFusion Markup Language (CFML) tags, HTML tags, Custom CFML tags, CFScript, JavaScript code, and anything else that an HTML page can contain" (see CF Web, page 7, first paragraph). When a browser client, such as Netscape Communicator, or Microsoft Internet Explorer, requests a ColdFusion application page, ColdFusion first processes the ColdFusion application page and outputs HTML code to the browser client (see CF Web, page 7, second paragraph). The HTML code output from the ColdFusion 4.0 software application is not an executable programming code, as claimed, because the HTML code needs further interpretation in order to be converted into executable programming code. This additional interpretation is done by the browser client – not by the ColdFusion 4.0 software product. Furthermore, all the processing of the ColdFusion application page into HTML code, and all subsequent interpreting of the HTML code is done at runtime. That is, in the ColdFusion 4.0 software product, there is no distinction between what events occur at translation time and what events occur at runtime.

The Examiner argues in the Advisory Action of May 3, 2004, that "one of ordinary skill in the computer art would recognize that any interpreting of code contained in a markup language document is equivalent to translating and executing said code," referring to the definitions of "interpret," "interpreted," and "interpreter" on page 285 of "Microsoft® Computer Dictionary," 2002, 5th ed., Microsoft Press. However, this is irrelevant, since no interpretation (at least not in accordance with the cited passages of the above-mentioned dictionary) is performed by ColdFusion. ColdFusion merely processes the ColdFusion application pages and outputs mark-up language code (e.g., HTML pages), which is in turn interpreted (i.e., translated into executable code at runtime) by the browser clients - **not** by ColdFusion itself, as alleged by the Examiner. Clearly, the processing described in CF Web does not teach translation of a page into executable programming code at translation time, which is executed at runtime to perform actions intended by the action tags. For at least these reasons, it is respectfully submitted that ColdFusion does not teach the translator of claims 1 and 13.

(c) The cited art does not teach a TagExtraInfo object

Claims 1 and 13 recite:

"a TagExtraInfo object for each action tag in the page, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method returning, at translation time, information that includes a list of available scripting variables, and a

variable type and scope associated with each scripting variable that is defined or modified by its associated action tag, thereby allowing the translator at translation time to use the information provided by the method to generate code that when executed at runtime will assign each of the scripting variables with appropriate runtime values with respect to the type and scope of each of the scripting variables."

The Examiner has asserted that the ColdFusion 4.0 software product teaches the TagExtraInfo object, and refers specifically to CF Advanced, page 27, subsection titled "Ancestor data access," which describes a 'GetBaseTagData' function that returns an object that contains all the variables, scopes, etc. of an n^{th} ancestor with a given name. The applicants respectfully disagree. As was discussed above, scripting variables is not one of the available variable types in ColdFusion. Furthermore, the "Ancestor data access" section of CF Advanced fails to teach a TagExtraInfo object for each action tag in a page. It should be noted that the TagExtraInfo object provides a method that is accessed by a translator at translation time. The method returns, at translation time, information that includes a list of available scripting variables, and a variable type and scope associated with each scripting variable that is defined or modified by its associated action tag. Thus, the method allows the translator, at translation time, to use the information provided by the method to generate code that, when executed at runtime, assigns each of the scripting variables with appropriate runtime values with respect to the type and scope of each of the scripting variables.

Clearly, the "Ancestor data access" section of CF Advanced fails to teach these features. Even from a very broad perspective, a function that returns "ancestor data" merely provides information about one or more ancestors. As such, a function that merely provides ancestor data does not provide a list of available scripting variables. Accordingly, even from a very broad perspective, the functions listed in the "ancestor data access" section CF Advanced cannot be used to provide a list of available scripting variables, and a variable type and scope associated with each scripting variable.

Moreover, the "Ancestor data access" of CF Advanced cannot be used by a translator that generates executable code, as required by claims 1 and 13. This is evident as there is no translator or translation of executable code in the CF Advanced environment, as was discussed above. It should also be noted that there is no motivation or suggestion to use the "ancestor data access," to provide the list of available scripting variables, notwithstanding the fact that such a combination may be technologically awkward. The applicants would also respectfully like to point out that the Examiner has not addressed the feature of a TagExtraInfo object for each action tag in a page. Instead, the Examiner has merely noted that a function (GetBaseTagList()) can be called to return a comma-delimited list of uppercased ancestor tag names (Final Office Action,

page 8, citing the fifth line of the code sample). Accordingly, the applicants submit that the ColdFusion 4.0 software product does not teach a TagExtraInfo object as required in claims 1 and 13.

(d) The cited art does not teach a pageContext object

Claims 1 and 13 recite:

> "a pageContext object for the page, the pageContext object including a runtime mapping of at least one scripting variable in the list of available scripting variables to a runtime value that is represented or can be represented in the tag library;"

The pageContext object provides a runtime mapping of at least one scripting variable in the list of available scripting variables. As noted above, the list of available scripting variables is provided (returned) by the method of the TagExtraInfo object at translation time. The Examiner has asserted that the ColdFusion 4.0 software product teaches the pageContext object, and refers specifically to CF Web, pages 16-17, and to CF Advanced, pages 26-27, respectively, that describe a CFSet instruction that can be used to create local variables. The applicants respectfully disagree and submit that contrary to the Examiner's assertion, the cited sections of CF Web and CF Advanced merely teach creating and using variables in the ColdFusion software environment. The tutorial does not teach providing a runtime mapping of at least one scripting variable in the list of available scripting variables that has been generated at translation time. The tutorial does also not make any mention of a tag library, which is required by the pageContext claim element. It should also be noted that the Examiner appears to have made a general allegation in the Final Office Action mailed January 5, 2004 (without providing a specific reference) that these features are taught by the sections "using Application and Session variables," "Using CGI Environment Variables," and "creating HTTP cookie variables." This general allegation does not address the claimed features, because the applicants have not attempted broadly claim using a variable. Nevertheless, the Examiner seems to have selected various sections of the ColdFusion art that merely mention the word "variable." Clearly, the Examiner has not addressed the pageContext object feature because, aside form the word "variable" in the title of the cited sections, the Examiner has not provided any factual basis to support the rejection. Accordingly, the applicants submit that the ColdFusion 4.0 software product does not teach a pageContext object as described in claims 1 and 13.

(e) The cited art does not teach a tag handler

Claims 1 and 13 recite:

> "a tag handler that creates at runtime one or more objects that the page requires, the tag handler further operating to store the one or more created objects into the pageContext object"

The tag handler stores at runtime one or more created objects into the pageContext object. This allows the one or more created objects to be retrieved at runtime, when the code that was generated at translation time is executed. The one or more created objects are assigned at runtime to the scripting variables in the list of scripting variables that is returned by the method in the pageContext object at translation time.

In the Final Office Action, the Examiner has asserted that these features are taught by ColdFusion 4.0 as evidenced by the "Dynamic Parameter" and " Expression" examples in CF Web (Final Office Action, page 9). The applicants respectfully disagree. Contrary to the Examiner's assertion, the cited examples merely demonstrate using typeless variables in ColdFusion. When a typeless variable is used, the variable type does not need to be identified. Accordingly, a variable can be based on a column name or an expression (CF Web, page 17). The typeless variables, however, do not create at runtime one or more objects that the page requires. Moreover, the typeless variables do not store the one or more created objects into another object (e.g., the pageContext object). Clearly, using a typeless variable does not address the claimed features the tag handler.

It should also be noted that the Examiner again has made general allegations that various sections of the cited art (the "Using Application and Session Variables," "Creating HTTP Cookie Variables," and "Using CGI Environment Variables" sections of CF Web, pages 36-44) somehow teach these features. This general allegation does not address the claimed features because, again, the applicants have not broadly attempted to claim using a variable. Nevertheless, also here the Examiner seems to have merely selected various sections of the art that make a reference to the word "variable." The applicants therefore submit that the Examiner has not addressed the tag handler feature because, aside form the word "variable" in the title of the cited sections, the Examiner has not provided any factual basis to support the rejection. Accordingly, the applicants submit that the ColdFusion 4.0 software product does not teach a tag handler as described in claims 1 and 13.

Therefore, for at least the reasons above, it is respectfully submitted that the ColdFusion 4.0 software product does not render the invention as currently recited in claims 1 and 13 unpatentable under 35 U.S.C. §102(a).

2. Independent claim 7

Independent claim 7 describes a method for automatically synchronizing scripting variable between a page including one or more action tags and a tag library. The method described in claim 7 contains limitations similar to the limitations of claim 1 and 13, and was thus cursorily rejected by the Examiner under 35 U.S.C. §102(a) as being anticipated by the ColdFusion 4.0 software product (Final Office Action, page 9, 3rd paragraph).

As was discussed above with respect to claims 1 and 13, the ColdFusion 4.0 software product does not teach automatic synchronization of scripting variables between a page including action tags and a tag library. Nor does the cited art teach translation of a page into an exectutable programming code. Nor does the cited art teach a TagExtraInfo object, a pageContext object, or a tag handler, or any combination thereof. All of these elements are required in claim 7. As a consequence of the lack of these components, the steps recited in claim 7 cannot occur in ColdFusion 4.0. For example, the "instantiating..." step requires "a list of available scripting variables and respectively associated variable types and scopes for each of the scripting variables in the list of available scripting variables, each of the scripting variables being defined or modified by an associated action tag." The ColdFusion software product merely provides ancestor information, and can thus only provide a list of available variables, all of which are related in an ancestral fashion.

The "receiving..." step requires that "a collection of returned TagExtraInfo objects" is received. No TagExtraInfo objects exist in the ColdFusion 4.0 software product.

The "generating..." step that requires a translator, TagExtraInfo objects, a pageContext object, and recites "generating...executable code that is executed at runtime, wherein the executable code accesses at runtime data that will be stored in a pageContext object at runtime." No such generation (at translation time) of executable code (to be executed at runtime) exists in ColdFusion 4.0. As was discussed above, the ColdFusion 4.0 software product merely outputs HTML code, which is later converted into executable code by a web browser. Furthermore, as was discussed above with respect to claim 1, no pageContext object exists in the ColdFusion 4.0 software product. The pageContext object is also required by the "storing..." step of claim 7,

which consequently cannot be anticipated by the ColdFusion 4.0 software product for at least this reason.

Finally, the "executing..." step requires both the pageContext object and the TagExtraInfo objects, and cannot be anticipated by the ColdFusion 4.0 software product for at least these reasons.

It ought to be clear from the above discussion, and without going into further detail of the claimed features of claim 7, that claim 7 is not anticipated under 35 U.S.C. §102(a) by the ColdFusion 4.0 software product. Thus the applicants submit that the rejection of claim 7 thus be withdrawn.

3. Dependent claims 4 and 14

Claims 4 and 14 depend directly from independent claims 1 and 13, respectively, and are therefore submitted to be patentable over the art of record for at least the reasons set forth above with respect to claims 1 and 13. Furthermore, dependent claims 4 and 14 recite the following features of the TagExtraInfo object:

> "the TagExtraInfo object comprises:
> a valid object name for each variable;
> a type for each variable; and
> a scope parameter that specifies a variable's scope relative to the page. "

The Examiner cites the same sections that were cited in the context of the TagExtraInfo object for claim 1 and 13. Again, the applicants respectfully disagree and contend that the "Ancestor data access" functions 'GetBaseTagList()' and 'GetBaseTagData' in the cited sections of CF Advanced, merely provide ancestor data. The 'GetBaseTagList' function returns a comma-delimited list of uppercased ancestor tag names, and the 'GetBaseTagData' function returns an object that contains all the variables, scopes, etc. of the nth ancestor with a given name. Neither of the functions returns "a valid object name for each variable;" "a type for each variable;" and "a scope parameter that specifies a variable's scope relative to the page" as required by claims 4 and 14.

Therefore, for at least these reasons, it is respectfully submitted that the ColdFusion 4.0 software product does not render the invention as currently recited in claims 4 and 14 unpatentable under 35 U.S.C. §102(a).

4. Dependent claim 8

Claim 8 depends directly from independent claim 7, and is therefore submitted to be patentable over the art of record for at least the reasons set forth above with respect to claim 7. Furthermore, dependent claim 8 recites the same features of the TagExtraInfo object as claims 4 and 14 discussed above.

Again, the applicants respectfully disagree and contend that the "Ancestor data access" functions 'GetBaseTagList()' and 'GetBaseTagData' in the cited sections of CF Advanced, merely provide ancestor data. The 'GetBaseTagList' function returns a comma-delimited list of uppercased ancestor tag names, and the 'GetBaseTagData' function returns an object that contains all the variables, scopes, etc. of the nth ancestor with a given name. Neither of the functions returns "a valid object name for each variable;" "a type for each variable;" and "a scope parameter that specifies a variable's scope relative to the page" as required by claim 8.

Therefore, for at least these reasons, it is respectfully submitted that the ColdFusion 4.0 software product does not render the invention as currently recited in claim 8 unpatentable under 35 U.S.C. §102(a).

5. Dependent claims 15 and 17

Claims 15 and 17 depend directly from independent claims 13 and 1, respectively, and are therefore submitted to be patentable over the art of record for at least the reasons set forth above with respect to claims 13 and 1. Furthermore, claims 15 and 17 require additional elements that when considered in the context of the claimed invention further distinguish the claimed invention from the cited art. More particularly, dependent claims 15 and 17 recite:

> "the page is converted to a first programming code which is different than a second programming code that is used to implement the tag library."

Thus, the page (for example, the JSP) is converted to a programming code, which is different from a programming code in which the tag library is implemented. In the Final Office Action, the Examiner cites CF Advanced, pages 9-10, as showing "an example of a JavaScript object in addition to the normal HTML-only page without making any changes to the tag library" (Final Office action, page 10, lines 3-4). The applicants respectfully disagree. The claim limitation in claims 15, 17 and 18 refers to a conversion of a page and to a tag library. The cited example, on the other hand, "demonstrates the transfer of a CFQUERY result set from a CFML template executing on the server to a JavaScript object that is processed by the browser." (CF

Advanced, page 9, lines 2-3). A CFQUERY result set is different from a page, which is required by the above claim limitation. The transfer from a template executing on the server to an object that is processed by the browser that is discussed in the cited section of CF Advanced is also irrelevant for the above claim limitation, as the applicants are not attempting to claim such a server-client transfer. Finally, no tag libraries exist in ColdFusion, as was discussed above.

Therefore, for at least the reasons above, it is respectfully submitted that ColdFusion 4.0 does not anticipate the invention as currently recited in claims 15 and 17. Accordingly, it is respectfully submitted that claims 15 and 17 are patentable over the cited art for at least these reasons and the reasons stated above with regards to independent claims 13 and 1.

6. Dependent claim 18

Claim 18 depends directly from independent claim 7 and is therefore submitted to be patentable over the art of record for at least the reasons set forth above with respect to claim 7. Furthermore, dependent claim 18 recites the same features as claims 15 and 17 discussed above.

Therefore, for at least the reasons presented above with respect to claim 7 and with respect to claims 15 and 17, it is respectfully submitted that ColdFusion 4.0 does not anticipate the invention as currently recited in claim 18. Accordingly, it is respectfully submitted that claim 18 is patentable over the cited art.

B) The rejection of claims 6, 10 and 16 under 35 USC § 103(a)

1. Claims 6 and 16

Claims 6 and 16 depend directly from independent claims 1 and 13, respectively, and are therefore submitted to be patentable over the ColdFusion 4.0 software product for at least the reasons set forth above with respect to claims 1 and 13. Furthermore, dependent claims 6 and 16 recite:

> "the page is executed on a server that implements a container, and the page is converted to a platform independent code that is executed on the server."

In the Final Office Action, the Examiner rejected claims 6 and 16 under 35 U.S.C. §103(a), stating that "it would have been obvious to one having ordinary skill in the computer art at the time the invention was made to substitute the known... JavaServerPage technology, in which a server implements a container, and a page is converted and executed on the server, for

the ColdFusion 4.0 technology. One would be motivated to do so because both are directed toward the same function."

The applicants respectfully disagree. The ColdFusion 4.0 software product cannot possibly disclose or reasonably suggest these features because it pertains to an environment where there is no need for automatic synchronization of scripting variables between a page and a tag library, and thus no need to convert the page into a platform independent code. Therefore, for at least these reasons, and the reasons discussed above with respect to claims 1 and 13, it is respectfully submitted that the ColdFusion 4.0 software product neither discloses nor reasonably suggests the invention as currently recited in claims 6 and 16. Accordingly, it is respectfully submitted that the rejection under 35 U.S.C. §103(a) of claims 6 and 16 be withdrawn.

2. Claim 10

Claim 10 depends directly from independent claim 7 and is therefore submitted to be patentable over the ColdFusion 4.0 software product for at least the reasons set forth above with respect to claim 7. Furthermore, dependent claim 10 recites the same features as claims 6 and 16 discussed above.

Therefore, for at least these reasons, and the reasons discussed above with respect to independent claim 7 and dependent claims 6 and 16, it is respectfully submitted that the ColdFusion 4.0 software product neither discloses nor reasonably suggests the invention as currently recited in claim 10. Accordingly, it is respectfully submitted that the rejection under 35 U.S.C. §103(a) of claim 10 be withdrawn.

C) Drawings

In the Final Office action, the Examiner requested a set of new corrected drawings, because the copy of FIG. 2, did not comply with the requirements of 37 C.F.R. §1.84, and FIG. 2 does not clearly and competently illustrate the aspects of the present invention as it is disclosed in the specification. In the response to the Final Office action, mailed by the applicants on April 5, 2004, a replacement sheet for FIG. 2 was submitted. The applicants believe that the revised FIG. 2 is in compliance with 37 C.F.R. §1.84 and clearly and competently illustrates the aspects of the present invention as it is disclosed in the specification, and submits that all the drawings are now in allowable condition.

D) The use of the trademarks JAVA and JAVASERVER in the Specification

In the advisory action, mailed on May 3, 2004, the Examiner requested further clarification as to how the applicants intend the Office to interpret recitation of the term "Java" in the instant specification.

The Examiner stated that:

> "If, however, Applicant maintains that JAVA should be considered as a trademark rather than as a generic name used in trade...'generic terminology', e.g. 'JAVA programming language' is necessary in order to properly describe (under 35 U.S.C. §112) the instant invention."

The applicants have adopted the Examiner's suggestions and submitted a supplemental amendment along with the original version of this Appeal Brief that was filed on September 23, 2004, addressing this particular issue. The applicants believe that the supplemental amendment fully addresses the trademark issues for the specification and submit that the specification is now in compliance with 35 U.S.C. §112. In the event that this amendment cannot be entered at this point, the applicants respectfully request that the amendment addressing the objection to the specification be entered upon reopening of prosecution of this case.

E) Claim rejections under 35 USC § 112

Claim 4 was rejected in the Final Office action for lacking antecedent basis, since claim 4 referred to claim 3, which had been canceled by the applicants. In the response to the Final Office action, the applicants amended claim 4 to depend from claim 1. The applicants submit that claim 4 is now in proper condition and that the rejection under 35 USC § 112 be removed.

F) Conclusion

In view of the forgoing, it is respectfully submitted that none of the pending claims are anticipated or reasonably suggested by the Cold Fusion 4.0 software product as described in the cited documents and that the Examiner's rejections of the pending claims were erroneous. Accordingly, it is respectfully requested that the pending rejections of all of the claims be reversed.

Respectfully Submitted,
BEYER WEAVER & THOMAS, LLP

Fredrik Mollborn
Reg. No. 48,587

P.O. Box 778
Berkeley, California 94704-0778
(650) 961-8300

VIII. CLAIMS APPENDIX

CLAIMS ON APPEAL

1. (Previously Presented) A computer system for automatic synchronization of scripting variables between a page including action tags and a tag library, the computer system comprising:

a page suitable for building an application with dynamic web content, the page including one or more action tags that are provided as text in a mark-up language;

a tag library;

a translator suitable for translating the action tags from the mark-up language to an executable programming code that is executed at runtime to perform actions intended by the action tags;

a TagExtraInfo object for each action tag in the page, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method returning, at translation time, information that includes a list of available scripting variables, and a variable type and scope associated with each scripting variable that is defined or modified by its associated action tag, thereby allowing the translator at translation time to use the information provided by the method to generate code that when executed at runtime will assign each of the scripting variables with appropriate runtime values with respect to the type and scope of each of the scripting variables;

a pageContext object for the page, the pageContext object including a runtime mapping of at least one scripting variable in the list of available scripting variables to a runtime value that is represented or can be represented in the tag library,

a tag handler that creates at runtime one or more objects that the page requires, the tag handler further operating to store the one or more created objects into the pageContext object;

thereby allowing the one or more objects to be retrieved at runtime when the generated code is executed, the one or more objects being assigned at runtime to each of the scripting variables in the list of scripting variables that is returned by the method at translation time.

2. (Canceled)

3. (Canceled)

4. (Previously Presented) The computer system of Claim 1, wherein the TagExtraInfo object comprises:

a valid object name for each variable;

a type for each variable; and

a scope parameter that specifies a variable's scope relative to the page.

5. (Canceled)

6. (Previously Presented) The computer system of Claim 1, wherein the page is executed on a server that implements a container, and the page is converted to a platform independent code that is executed on the server.

7. (Previously Presented) A method for automatically synchronizing scripting variable between a page including one or more action tags and a tag library, the page suitable for building an application with dynamic web content, the one or more action tags being provided as text in a mark-up language which are translated at translation time to an executable code that is executed at runtime to synchronize the scripting variables at runtime, the method comprising:

instantiating, by a translator at translation time, for each action tag a TagExtraInfo object, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method capable of returning at translation time information that includes a list of available scripting variables and respectively associated variable types and scopes for each of the scripting variables in the list of available scripting variables, each of the scripting variables being defined or modified by an associated action tag;

invoking the method by a translator at translation time, wherein the invoking operates to pass a list of attributes associated with the one or more action tags;

receiving, as a result of the invoking of the method, a collection of returned TagExtraInfo objects, wherein each of the returned TagExtraInfo objects includes an available scripting variable, its variable type, and its scope in the page;

generating by the translator, based on the returned TagExtraInfo objects, executable code that is executed at runtime, wherein the executable code accesses at runtime data that will be stored in a pageContext object at runtime, the runtime data including appropriate runtime values for each of the available scripting variables;

storing at runtime, into the pageContext object, by a tag handler at runtime, one or more objects that the page requires, thereby allowing the objects for the one or more objects to be retrieved at runtime and be assigned at runtime to the list of scripting variables; and

executing, at run time, the code generated by the translator to assign appropriate runtime values which are stored in the pageContext object to each of the scripting variables in the returned TagExtraInfo objects, thereby allowing the runtime values to be retrieved and assigned at runtime to each of the available scripting variables in the collection of returned TagExtraInfo objects.

8. (Previously Presented) The method of Claim 7, wherein the TagExtraInfo object comprises:

a valid object name for each variable;

a type for each variable; and

a scope parameter that specifies a variable's scope relative to the page.

9. (Canceled)

10. (Previously Presented) The method of Claim 7 , wherein the page is executed on a server that implements a container, and the page is converted to platform independent code that is executed on the server.

11. (Canceled)

12. (Canceled)

13. (Previously Presented) A computer readable media including computer program code for automatically synchronizing scripting variables between a page including one or more action tags and a tag library, the computer readable media comprising:

computer program code for a page suitable for building an application with dynamic web content, the page including one or more action tags that are provided as text in a mark-up language;

computer program code for a tag library;

computer program code for a translator suitable for translating the action tags from the mark-up language to an executable programming code that is executed at runtime to perform actions intended by the action tags;

computer program code for a TagExtraInfo object for each action tag in the page, the TagExtraInfo object providing a method that is accessed by the translator at translation time, the method returning, at translation time, information including a list of available scripting variables, and a variable type and scope associated with each scripting variable that is defined or modified by its associated action tag; thereby allowing the translator, at translation time, to use the list to generate code that when executed at runtime will assign each of the scripting variables with appropriate runtime values with respect to the type and scope of each of the scripting variables;

computer program code for a pageContext object for the page, the pageContext object including a runtime mapping of at least one scripting variable in the list of available scripting variables to a runtime value that is represented or can be represented in the tag library; and

computer program code for a tag handler that creates at runtime one or more objects that the page requires, the tag handler further operating to store the one or more created objects into the pageContext object, thereby allowing the one or more objects to be retrieved at runtime when the code is executed, and the one ore more objects being assigned at runtime to each of the scripting variables in the list of scripting variables that was returned by the method at translation time.

14. (Previously Presented) A computer readable medium as recited in claim 13, wherein the TagExtraInfo object comprises:

a valid object name for each variable;

a type for each variable; and

a scope parameter that specifies a variable's scope relative to the page.

15. (Previously Presented) A computer readable medium as recited in claim 13, wherein the page is converted to a first programming code which is different than a second programming code that is used to implement the tag library.

16. (Previously Presented) A computer readable medium as recited in claim 13, wherein the page is executed on a server that implements a container, and the page is converted to a platform independent code that is executed on the server.

17. (Previously Presented) A computer system as recited in claim 1, wherein the page is converted to a first programming code which is different than a second programming code that is used to implement the tag library.

18. (Previously Presented) A method as recited in claim 7, wherein the page is written in a first programming code which is different than a second programming code that is used to implement the tag library.